System Enhancement

Data stream approach

Fri, Dec 28, 2001

One generic method of adding support for new types of data to a system is by using a local application to generate the data, defining a data stream to house the data, then using the data stream listypes to access the data. This note describes how this can be done.

The number of listypes is about 90-odd at this time. From experience, the listype table is seldom enlarged; the latest significant additions pertained to support for raw floating point analog data values. Many new needs for system support can be completely handled via a local application, which makes analog or digital data available via the normal listypes.

Suppose one used a local application to build some arbitrary data structures that are to be accessed via the data request mechanism. the local application can call `DSWrite` to record the data into a data stream, which would have to be defined via a DSTRM table entry so that its queue would be initialized at system reset time. Any interested client could then issue a data request to read out the contents of this queue, making us of the three listypes that support access to data stream queues.

As a review, the three listypes are as follows:

Listype	*Period*	*Meaning*
50	<= 15 Hz	Supply data records as of request initialization.
51	one-shot	Supply the most recently-recorded records.
78	any	Supply data records starting with oldest
52	any	Access data stream queue header (read only)
53	any	Access DSTRM table entry
54	any	Access DSTRM entry 8-character name

From experience, listypes 50 and 51 are most commonly used.

When defining a new DSTRM entry, it is normally done via memory access, although listype 53 could be used for this purpose. The structure of a 32-byte DSTRM entry is as follows:

Field	*Size*	*Meaning*
`QType`	2	Queue type should be 1, the only type supported.
`ESize`	2	Queue record size in bytes. For variable size, use 0.
`USize`	2	Size of user header component.
`QSize`	2	Size of queue, including header
`QPtr`	4	Address of queue.
`spare`	4	n.u.
`QName`	8	Name of data stream queue
`spare`	8	n.u.

To indicate that a data stream queue should be dynamically allocated at reset time, rather than refer to a statically-selected memory area, add `0x4000` into `QType` field. In any case, after successful queue initialization, the sign bit of `QType` will be set, enabling writing records into queue. In practice, one normally finds this field has the value `0x8001`. Also, in most cases, data streams are defined for fixed-size records, not variable size.

When a data stream queue has been initialized, the queue header appears as follows:

Field	*Size*	*Meaning*
`QTyp`	2	Queue type is always 1.
`ESiz`	2	Queue record size in bytes, copied from `ESize`.
`UOff`	2	Offset to header user area of size `USize`
`QSiz`	2	Size of queue, copied from `QSize`
`Total`	4	Count of records ever written into queue
`spare`	4	n.u.
`inOff`	2	Offset where next record is to be written
`limOff`	2	Maximum offset, copied from `QSize`
`stOff`	2	Start offset, to which `inOff` "wraps" on reaching `limOff`
`spare`	2	n.u.
`UArea`	(`USize`)	May be used for any purpose by user of queue
`QRec`	(`QSiz`)–(`stOff`)	Space for `((QSiz)–(stOff))/(ESiz)` records, assuming fixed record sizes.

The space in the queue to be used for records is cleared initially, to aid debugging.

When requesting data stream records, the specified buffer size must allow for a 4-byte header that is formatted as follows:

Field	*Size*	*Meaning*
`nRec`	2	Number of records included in this reply buffer
`rSize`	2	Record size, copied from `ESiz`.

If fixed size records are used, the rest of the reply buffer will be an array of records, with the number indicated by `nRec`.

If variable size records are used, the first 2-byte word of each record always contains the size of the record that follows. (Maybe the record size includes the 2-byte size word?)

Data streams can be accessed simultaneously by any number of clients. When a client reads from the queue, that act does not consume the records read, except for that client in the case of periodic replies. Separate clients do not interfere with each other.

There is no attempt made to freeze a data stream that is full; in fact, there is no concept of a "full" data stream queue. Each client must be aware of the maximum average rate of writing records into the queue in order to choose the proper buffer size and reply period to "keep up." When there are no records included in a reply to a periodic request, the `nRec` field will be zero.

Each data stream has a name, but data requests require an ident that contains the index into the DSTRM table for the data stream of interest. There should probably be a new listype defined that can return the index in response to a specified name ident. Without this, one has to "just know" the correct DSTRM table index. Alternatively, one could read out the entire DSTRM table (at least the name fields) and perform a search, which seems tedious unless one is executing a local application inside the node containing the data stream.